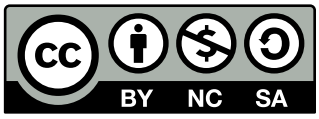
Ensemble Learning

Luis Norberto Zúñiga Morales

25 de febrero de 2024

Índice



Resumen

Ensemble Learning es un paradigma del aprendizaje automático cuyo principal objetivo es utilizar modelos de aprendizaje débiles en uno fuerte. Existen distintos modelos y algoritmos, a tal punto que, después de algunos años de su introducción, no se comprendía por qué funcionaban. Sin embargo, los métodos de ensamble suelen ser populares dado su habilidad para reducir sesgo o varianza.

1. Ensemble Learning

El *Ensemble Learning* (EL) es un paradigma de aprendizaje automático (*Machine Learning*) donde múltiples modelos de aprendizaje se entrenan para resolver un problema. En conjunto, los modelos intentan construir múltiples hipótesis para resolver el problema de aprendizaje, en lugar de una única hipótesis si se trabajara con un modelo en específico [16]. La idea básica del EL surge al imitar el comportamiento de aprendizaje social humano donde se busca el consejo u opinión de otros miembros de nuestro entorno social para realizar una decisión crucial o importante. De esta manera, se mezclan diferentes corrientes de pensamiento o experiencia de los expertos que se consultan para mejorar la toma de decisiones. De forma similar, el EL combina distintos modelos de aprendizaje (llamados modelos base) que combinan sus formas de aprendizaje, la forma en la que manejan los datos u otras características particulares para obtener mejores resultados en problemas de aprendizaje supervisado o no supervisado [4].

Al considerar el error de un modelo de regresión o clasificación como una mezcla de sesgo y varianza de la forma

$$E = \text{Sesgo}^2 + \text{Var} + \sigma^2$$

uno se preguntaría como es posible reducirlo, ya sea reduciendo o manteniendo la complejidad del modelo (sesgo) y reduciendo la varianza. Oportunamente, el EL ataca el problema anterior de una forma relativamente sencilla, ya sea manipulando el conjunto de datos o los modelos de aprendizaje empleados.

Aunque no existe una forma única de clasificar los distintos métodos que existen dentro del paradigma del EL, frecuentemente se observan dos grandes familias:

- **Métodos que promedian** (*Averaging Methods*), los cuales construyen distintos modelos de clasificación independientes entre ellos para promediar las predicciones que realizan. En esta clase entran los métodos de Bagging, Árboles y Bosques Aleatorios, entre otros.
- **Métodos que aumentan** (*Boosting Methods*), los cuales combinan diferentes modelos débiles para que, en conjunto, puedan formar un modelo robusto reduciendo el error de entrenamiento. En esta clase entran modelos como AdaBoost, Gradient Boosting, XGBoosting, entre otros.

La clasificación anterior es una entre muchas. Cabe resaltar otras como Bagging vs Boosting, o la propuesta por Re y Valentini [13] donde consideran dos grandes clases: métodos generativos y no generativos, donde la principal diferencia consiste en si generan o no nuevos modelos base como parte del proceso del ensamble. Otra clasificación incluye los modelos que votan, los cuales incluyen Bagging, Boosting y variantes.

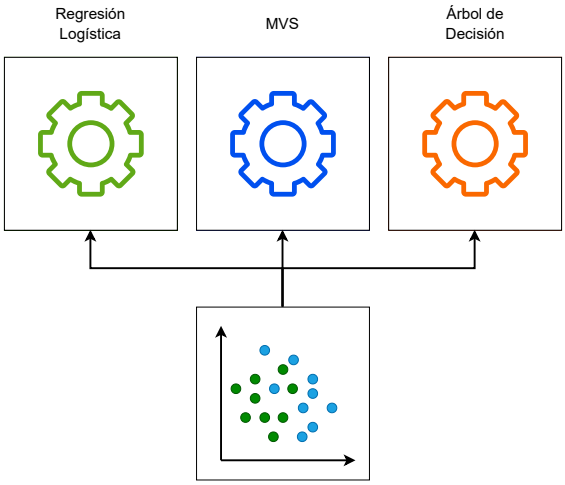


Figura 1: Idea de un ensamble de modelos de clasificación.

2. Voting Classifiers

Supongamos que durante la fase de elección de modelos para un problema de clasificación, se optó por comparar el rendimiento de cada uno de ellos para elegir el mejor. Sin ninguna razón en específico para la elección de cada modelo, se limitó a usar regresión logística, una Máquina de Vectores de Soport y un Árbol de Decisiones, como lo demuestra la Fig. (1).

Una manera simple de obtener un mejor sistema de predicciones, es combinar las predicciones de cada modelo base y predecir la clase de cada instancia del conjunto de datos por medio de una votación como lo indica la Fig. (2). La clase que tenga más votos es asignada a la instancia en turno. Este tipo de votación se le conoce como votación dura (*hard voting*).

Dicho de otra forma, supongamos que se tienen M distintos modelos de aprendizaje $\{\phi_m \,|m = 1, ..., M\}$ todos utilizando el conjunto de datos L. El ensamble, representado como $\psi_{\phi_1,...,\phi_M}$, busca reducir el error de generalización al considerar el error esperado de cada modelo individual.

Para el caso de clasificación, la predicción se puede hacer de dos formas. La votación dura se expresa de la siguiente manera:

$$\psi_{\phi_1,...,\phi_M}(\boldsymbol{x}) = \underset{c \in y}{\operatorname{argmax}} \sum_{m=1}^{M} 1(\phi_m(\boldsymbol{x}) = c) \tag{1}$$

Otra manera para determinar el voto mayoritario se denomina voto suave (*soft voting*). Cuando algún modelo de aprendizaje individualmente estima una probabilidad $\hat{p}_L(Y = c|X = \boldsymbol{x})$ para cada clase, es posible promediar la probabilidad estimada para cada una de ellas y después asignar aquella que sea la más probable:

$$\psi_{\phi_1,...,\phi_M}(\boldsymbol{x}) = \underset{c \in y}{\operatorname{argmax}} \frac{1}{M} \sum_{m=1}^{M} \hat{p}_L(Y = c|X = \boldsymbol{x}) \tag{2}$$

La razón por la que el voto mayoritario funciona se puede explicar gracias al Principio del Jurado de Condorcet.

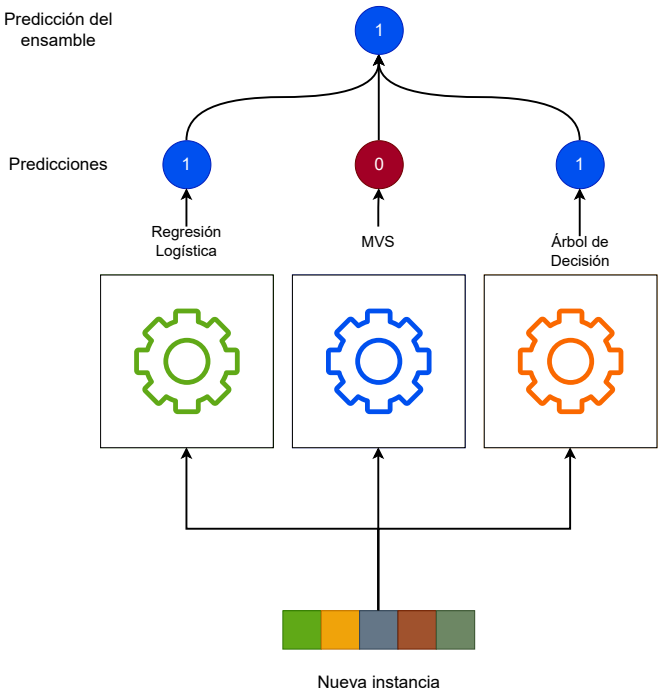


Figura 2: Votación dura para un ensamble de modelos de aprendizaje que predicen una nueva instancia del conjunto de datos.

Teorema 1 (Principio del Jurado de Condorcet)**.** *Suponga que se tiene un jurado compuesto por M personas las cuales desean llegar a un veredicto por medio de una mayoría en una votación. Si cada jurado tiene una probabilidad independiente $p > \frac{1}{2}$ de elegir la decisión correcta, añadir más votantes incrementa la probabilidad de que la decisión colectiva sea la correcta. Cuando $M \to \infty$, la probabilidad de que se tome la decisión correcta tiende a 1. Por otro lado, si $p < \frac{1}{2}$, cada votante es más propenso a votar de forma incorrecta e incrementar M empeora las cosas.*

Supongamos que tenemos una moneda ligeramente cargada con una proporción 51/49 de que salga cara o cruz, respectivamente. Si se realiza el experimento de lanzar la moneda 1000 veces se obtendrán, aproximadamente, 510 caras y 490 cruces, una mayoría de caras. Para un poco más de 1000 lanzamientos, la probabilidad de obtener una mayoría de caras es cercana al 75 %, debido a la ley de los grandes números. De forma similar, supongamos que se construye un ensamble de 1000 clasificadores donde el 51 % predicen valores correctos de forma individual. Si lo que predicen correctamente es la clase más votada en el ensamble, se puede esperar una precisión del 75 % al predecir la clase correcta. Sin embargo, eso solo sucede si los modelos de clasificación empleados en el ensamble son independientes unos de otros, lo cual implica que los errores que cometen son independientes entre ellos. Lo anterior es difícil de evitar ya que todos se entrenan con los mismos datos, aunque una forma de reducir esta independencia es entrenar modelos que utilicen distintos algoritmos para su entrenamiento.

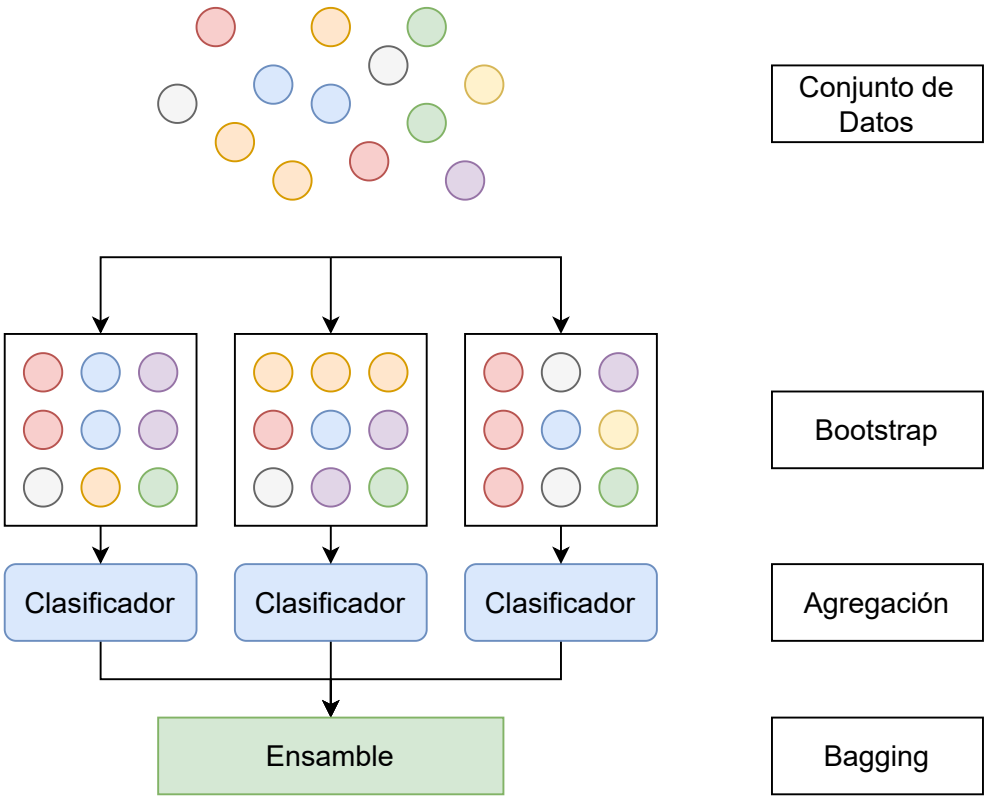


Figura 3: Diagrama que ejemplifica el concepto del Bagging.

3. Bagging

Bagging, acrónimo de *Bootstrap Aggregation*, es un método de EL que busca reducir la varianza en conjuntos de datos ruidosos y evitar el sobreajuste que históricamente se ha observado en árboles de decisión. Al practicar el Bagging [1], se eligen varias muestras de datos con remplazo[1] del conjunto de entrenamiento. Después, cada muestra se entrena de forma independiente y, dependiendo de la tarea (regresión o clasificación), se elige el promedio o mayoría de las predicciones.

Un conjunto de entrenamiento L consiste de puntos de datos $\{(\boldsymbol{x}_i, y_i), i = 1, ..., N\}$, donde $\boldsymbol{x}_i$ es el vector de características que representa a cada punto de datos y y_i es la etiqueta de la clase o valor numérico asociado al vector de características. Supongamos que se emplea un modelo de aprendizaje para encontrar un función predictora $f(\boldsymbol{x}, L)$ que toma como entrada el vector de características $\boldsymbol{x}$ tal que $y = f(\boldsymbol{x}, L)$. Ahora, supongamos que se tiene una secuencia de conjuntos de entrenamiento $\{L_k\}$ cada uno con N observaciones independientes del conjunto original L. El objetivo del Bagging es usar $\{L_k\}$ para obtener una mejor función predictora $\tilde{f}(\boldsymbol{x}, L_k)$ mediante la construcción de nuevas funciones $\{\phi_{L_k} \ | k = 1, ..., M\}$ que forman el ensamble $\psi(\boldsymbol{x})$.

Para el caso de un problema de clasificación, se puede considerar el voto duro expresado en la Ecuación 1; o el voto suave definido en la Ecuación 2. Para el caso de regresión, se

[1]Los puntos elegidos de la muestra se regresan a la población, por lo que pueden ser elegidos más de una vez durante el proceso de selección.

puede optar por un promedio de los valores numéricos que arroja el modelo de aprendizaje:

$$\psi(\boldsymbol{x}_i) = \frac{1}{M}\sum_{k=1}^{M}\phi_{L_k}(\boldsymbol{x}_i) \tag{3}$$

Por ejemplo, sea $L = \{1, 2, 3, 4, 5, 6\}$. Al realizar el muestro con reemplazo, se pueden obtener los siguientes subconjuntos para entrenar nuevos modelos de aprendizaje:

$$L_1 = 1, 2, 2, 3, 4, 1$$
$$L_2 = 1, 5, 6, 2, 2, 3$$
$$L_3 = 2, 3, 1, 5, 6, 6$$

Después, se crean nuevos modelos independientes $\phi_{L_1}, \phi_{L_2}, \phi_{L_3}$ que se entrenan por separado y se someten a un voto duro o suave para el caso de clasificación, o un promedio para el caso de regresión.

La idea de generar nuevos subconjuntos de datos a partir del conjunto de datos original tiene propiedades atractivas desde el punto de vista estadístico. Por ejemplo, las muestras se obtienen de una fuente que, en teoría, representa la realidad y sigue la distribución de probabilidad que rige el fenómeno observado. Además, estas muestras son independientes entre ellas, lo cual lleva a suponer que son muestras independientes e idénticamente distribuidas. Sin embargo, para que tal propiedad se cumpla, el conjunto de datos original debe ser lo suficientemente grande para que: a) pueda capturar la complejidad de la distribución de probabilidad y el nuevo muestreo pueda equipararse a uno de la distribución original, y b) para que las muestras no presenten correlación entre ellas y el conjunto de datos. En resumen, se debe lograr un balance entre representatividad e independencia.

En 1996, Breiman [1] introdujo el algoritmo para realizar Bagging. En general, consta de las siguientes etapas:

1. **Bootstrapping**: se realizan múltiples muestras o subconjuntos del conjunto de entrenamiento. Dichas muestras se realizan mediante muestreo con reemplazo.

2. **Entrenamiento**: cada muestra o subconjunto se entrena de forma paralela e independiente entre ellas, ya sea con modelos débiles o modelos base.

3. **Agregación**: Según la tarea (regresión o clasificación), se toma un promedio o la mayoría de las predicciones para calcular una estimación más precisa.

En el mismo artículo, Breiman menciona el siguiente procedimiento para un modelo de aprendizaje mediante Árboles de Decisión:

1. Divide el conjunto de datos en un conjunto de entrenamiento L y uno de prueba T, en una proporción 9:1.

2. Se construye un árbol de decisión con el conjunto de entrenamiento L utilizando validación cruzada con 10 pliegues. Al final, se obtiene un error $e_S(L, T)$ al predecir valores con el conjunto de prueba T.

3. Una muestra L_B se crea mediante Bootstrap del conjunto L y se entrena un nuevo árbol usando L_B. El conjunto de entrenamiento original L se usa como conjunto de prueba para seleccionar el mejor árbol podado. Esto se repite 50 veces, obteniendo 50 árboles diferentes $\phi_1(\boldsymbol{x_i}), ..., \phi_{50}(\boldsymbol{x_i})$.

4. Si $(\boldsymbol{x_i}, j_i) \in T$, se sigue que la clase estimada de $\boldsymbol{x_n}$ es aquella que tenga la mayor cantidad de votos entre todos los árboles $\phi_1(\boldsymbol{x_i}), ..., \phi_{50}(\boldsymbol{x_i})$. Si existe un empate, la clase estimada es la que tenga la menor etiqueta de clase. La razón de veces que estima de manera errónea se conoce como la razón de mala clasificación $e_B(L, T)$.

5. La división aleatoria del conjunto de datos en L y T se repite 100 veces y se calcula el promedio de e_S y e_B.

Además, Breiman menciona que usar Bagging es efectivo en modelos de aprendizaje inestables, donde pequeños cambios en el conjunto de datos causan grandes cambios en las predicciones del modelo. Ejemplos de dichos modelos inestables incluyen Árboles de Decisión, Redes Neuronales Artificiales y algunos modelos de Regresión Lineal.

Dentro de las ventajas que supone utilizar Bagging se encuentran las siguientes:

- Reduce el factor de la varianza cuando un modelo de aprendizaje débil presenta bajo sesgo y alta varianza.
- El ensamble mejora el rendimiento de los modelos base débiles.
- Se puede realizar el entrenamiento en paralelo.

Sin embargo, también presenta algunas desventajas:

- Para modelos de aprendizaje débiles con alto sesgo, al emplear Bagging se transfiere el alto sesgo. En general, el Bagging no reduce el sesgo, únicamente la varianza.
- Pérdida de la interpretabilidad de los modelos de aprendizaje empleados.
- Puede ser complejo computacionalmente, ya que los subconjuntos pueden llegar a ser de gran tamaño, dependiendo del conjunto de datos base que se utilice.

4. Pasting

En 1999, Breiman [2] introdujo la idea de *Pasting* como un modelo de EL. Consideremos un conjunto de datos L compuesto de ejemplos $(\boldsymbol{x}_i, y_i)$, donde $\boldsymbol{x}_i$ es el vector de características y y_i la clase o valor asignada a dicho vector, se utiliza para entrenar un modelo de aprendizaje, pero el conjunto L es muy grande como para que la memoria de la computadora lo pueda manejar. La idea del Pasting es crear muestras de menor tamaño de L que sirven de entrada a distintos modelos de aprendizaje agregándolos para obtener resultados cercanos al óptimo.

La idea del Pasting se conforma de dos puntos claves:

- Supongamos que hasta el momento se han construido k predictores. Un nuevo conjunto de entrenamiento de tamaño N se selecciona de L ya sea por muestreo aleatorio (usualmente sin reemplazo) o de importancia. El $(k+1)$-ésimo predictor se entrena en este nuevo conjunto de entrenamiento y se agrega con los k anteriores. La agregación se realiza por voto mayoritario (sin pesos). Si se utiliza un muestreo aleatorio para seleccionar el conjunto de entrenamiento, este se llama *Rprecinct* y el procedimiento se llama *pasting Rvotes* (R = aleatorio). Si se realiza mediante muestreo por importancia, el conjunto de entrenamiento se llama *Iprecint* y el procedimiento, *pasting Ivotes* (I = importancia). En el caso donde se considera la importancia, el muestreo le da mayor peso a aquellos puntos que tienden a ser mal clasficados.

- Se actualiza una estimación $e(k)$ del error de generalización para la k-ésima agregación $e(k)$. El pasting se detiene cuando $e(k)$ deja de disminuir. La estimación de $e(k)$ se puede obtener usando un conjunto de prueba fijo, aunque también puede usarse el conjunto fuera de la bolsa (*out-of-bag*).

Tanto en el Bagging como en el Pasting, algunas instancias pueden repetirse múltiples veces durante el muestreo al momento de crear los subconjuntos que se utilizan para entrenar los nuevos modelos del ensamble. Es posible que otros puntos no lleguen a ser seleccionados ni una sola vez durante el proceso. Durante el Bagging, que considera muestreo con reemplazo, alrededor del 63 % de las instancias del conjunto de entrenamiento son seleccionadas durante el muestreo. El restante 37 % de las instancias que no se eligen conforman el conjunto fuera de la bolsa (*out-of-bag*), los cuales no son necesariamente los mismos para todos los modelos de aprendizaje.

5. Boosting

Boosting es una familia de métodos de EL que combina un conjunto de modelos de aprendizaje débiles en un modelo fuerte para reducir el error que se presenta durante el entrenamiento. A diferencia del Bagging, donde los modelos de aprendizaje se entrenan en paralelo, el Boosting selecciona muestras aleatorias las cuales se utilizan para ajustar una serie de modelos de aprendizaje homogéneos en forma secuencial. Con cada iteración, las reglas débiles de cada clasificador individual se combinan para forma una única regla fuerte. Las instancias que se predicen incorrectamente son elegidas en cada nueva iteración con mayor frecuencia en lugar de aquellas que se predicen correctamente y se construyen nuevos modelos que buscan ser mejores que sus antecesores.

Por ejemplo, supongamos que se tienen 6 instancias de entrenamiento $L = \{1, 2, 3, 4, 5, 6\}$ donde la instancia 1 presenta dificultades al modelo de clasificación. A diferencia del Bagging, donde cada nuevo conjunto se construye de forma independiente, el Boosting aumenta la frecuencia de la clase difícil de predecir, concentrándose en ella para predecirla de forma correcta.

$$L_1 = 1, 2, 2, 3, 4, 6$$
$$L_2 = 1, 5, 6, 1, 2, 3$$
$$L_3 = 1, 1, 1, 5, 4, 2$$

En resumen, el Boosting intentar producir nuevos clasificadores capaces de mejorar el ensamble mediante una mejora al rendimiento en la predicción de instancias históricamente mal asignadas, esto es, aquellos modelos que se encuentran ligeramente por arriba del baseline aleatorio.

El Boosting permite reducir el sesgo presente en los modelos base, por lo que es conveniente utilizar aquellos que presenten alto sesgo y baja varianza. Además, dada la naturaleza secuencial del algoritmo, utilizar modelos con alto sesgo y baja varianza son menos costosos computacionalmente dado un menor grado de libertad en ellos al momento de ser parametrizados.

El siguiente punto a considerar, ya que explicamos los puntos básicos que dan forma a la idea del Boosting, es cómo realizar las muestras con importancia, la forma en la que se ajustan los modelos y cómo se agregan. Por el momento, vamos a considerar dos opciones: AdaBoost y Gradient Boosting.

5.1. AdaBoost

En 1995 Freund y Schapire [5, 6] introdujeron el algoritmo de Boosting denominado AdaBoost (*Adaptative Boosting*). La idea básica es la siguiente: el algoritmo entrena un clasificador base y lo utiliza para hacer predicciones en el conjunto de datos base. Después, el algoritmo le asigna un mayor peso a los puntos mal clasificados, aumentando su importancia. En la siguiente iteración, se entrena un nuevo modelo base con los pesos actualizados del conjunto de datos, realiza sus predicciones y vuelve a actualizar los pesos.

Para la derivación del modelo, utilizaremos el desarrollo propuesto por Rojas [14]. Supongamos que se tiene un conjunto de datos L y, para cada punto $\boldsymbol{x}_i$, cada modelo base del ensamble emite una opinión $\phi_i(\boldsymbol{x}_i) \in \{-1, 1\}$. Al final, el ensamble, formado por M modelos, toma una decisión dada por $\text{sign}(C(\boldsymbol{x}_i))$, donde $C(\boldsymbol{x}_i)$ es una suma ponderada:

$$C(\boldsymbol{x}_i) = \sum_{j=1}^{M} \alpha_j \phi_j(\boldsymbol{x}_i) \tag{4}$$

Las constates α_i son los pesos que se le asignan a cada modelo base, y la salida de cada uno de ellos puede ser -1 o 1. La idea de este modelo de Boosting reside en elegir el mejor modelo de aprendizaje en cada iteración por medio del cálculo del error de clasificación y asignar el peso α_k respectivo que denota la importancia que tiene el clasificador en el ensamble. En general, el algoritmo de AdaBoosting consiste de tres pasos:

1. Explorar los posibles clasificadores para agregar al ensamble.
2. Seleccionar el clasificador que se añade al ensamble.
3. Determinar el peso α_k para el nuevo miembro y asignar la importancia a los miembros del conjunto de datos.

5.1.1. Exploración

El primer paso consiste en probar los clasificadores del ensamble por medio de un conjunto de entrenamiento T de N puntos $(\boldsymbol{x}_i, y_i)$. Para realizar la prueba, a cada modelo se le asigna un costo e^{β} cada vez que el clasificador falla, y $e^{-\beta}$ si el clasificador predice la clase correcta para la instancia $\boldsymbol{x}_i$. AdaBoost requiere que $\beta > 0$, lo que ocasiona que los fallos se penalicen más que los aciertos. A este tipo de error (e^{β}) se le conoce como función de pérdida exponencial.

Cuando se prueban los k clasificadores, se construye una matriz S que alberga los resultados de aciertos (0) y fallos (1) de cada uno de ellos. Por ejemplo, en la matriz representada en la Fig. (1) se aprecia una posible evaluación de los k modelos usados hasta el momento.

	ϕ_1	ϕ_2	...	ϕ_k
$\boldsymbol{x}_1$	0	1	...	1
$\boldsymbol{x}_2$	0	1	..	0
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_k$	1	1	...	0

Cuadro 1: Matriz S que muestra aciertos y fallos de cada posible modelo candidato.

La idea del AdaBoost es extraer una clasificador de la lista en cada una de las M iteraciones. Los elementos del conjunto de datos reciben cierto peso según su relevancia en cada iteración. Al principio, a todos los elementos se les asigna el mismo peso, el cual puede ser 1 o $1/N$. Conforme avanza el proceso, a los puntos más complicados de clasificar se les asignan pesos cada vez mayores. Después, sigue un proceso de selección de nuevos clasificadores que permitan clasificar las clases mal asignadas. No es buena idea seguir empleando un modelo cuya opinión en cada ciclo sea errónea.

5.1.2. Selección del Nuevo Clasificador

En cada iteración, se necesita jerarquizar todos los posibles clasificadores para elegir al mejor de ellos. En la m-ésima iteración, ya se han incluido $m-1$ clasificadores en el ensamble, por lo que en este paso se debe agregar uno más. Hasta el momento, se tiene la siguiente combinación lineal:

$$C_{m-1}(\boldsymbol{x}_i) = \alpha_1\phi_1(\boldsymbol{x}_i) + ... + \alpha_{m-1}\phi_{m-1}(\boldsymbol{x}_i) \tag{5}$$

y se desea llegar a

$$C_m(\boldsymbol{x}_i) = C_{m-1}(\boldsymbol{x}_i) + \alpha_m\phi_m(\boldsymbol{x}_i) \tag{6}$$

Noten que, en la primera iteración, $C_{m-1} = 0$. Ahora, el error total del clasificador extendido se define como la pérdida exponencial:

$$E = \sum_{i=1}^{N} e^{-y_i(C_{m-1}(\boldsymbol{x}_i)+\alpha_m\phi_m(\boldsymbol{x}_i))} \tag{7}$$

donde los valores óptimos de α_m y ϕ_m se encuentran pendientes de ser encontrados. La Ecuación 7 se puede expresar de la siguiente forma:

$$E = \sum_{i=1}^{N} w_i^{(m)} e^{-y_i \alpha_m \phi_m(\boldsymbol{x}_i)} \tag{8}$$

donde

$$w_i^{(m)} = e^{-y_i C_{m-1}(\boldsymbol{x}_i)} \tag{9}$$

para $i = 1, ..., N$. En la primera iteración, $w_i^{(1)} = 1$. Posteriormente, para cada nueva iteración, el vector $w^{(m)}$ representa el peso asignado a cada dato en el conjunto de entrenamiento en la m-ésima iteración. La suma en la Ecuación 8 se puede dividir de la siguiente manera:

$$E = \sum_{y_i = \phi_m(\boldsymbol{x}_i)} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq \phi_m(\boldsymbol{x}_i)} w_i^{(m)} e^{\alpha_m} \tag{10}$$

lo cual nos indica que el error total es el error ponderado de los aciertos ($e^{-\beta}$) y el error ponderado de los fallos (e^{β}). La primera sumatoria se puede representar como $W_c e^{-\alpha_m}$ y la segunda como $W_e e^{\alpha_m}$:

$$E = W_c e^{-\alpha_m} + W_e e^{\alpha_m} \tag{11}$$

Para la selección de ϕ_m, el valor exacto de $\alpha_m > 0$ es irrelevante, ya que para α_m fija, minimizar E es equivalente a minimizar $e^{\alpha_m} E$ y por que

$$e^{\alpha_m} E = Wc + W_e e^{2\alpha_m} \tag{12}$$

Dado que $e^{2\alpha_m} > 1$, la Ecuación 12 se puede reescribir como

$$e^{\alpha_m} E = (W_c + W_e) + W_e(e^{2\alpha_m} - 1) \tag{13}$$

Noten que $(W_c + W_e)$ es la suma total de todos los pesos de los datos en el conjunto de datos, una constante en la iteración en turno. El segundo término de la Ecuación 13 se minimiza cuando en la m-ésima iteración se elige el clasificador con el menor error total W_e, i.e., el que tenga la menor proporción del error ponderado. En pocas palabras, tiene sentido elegir al que menor error tenga dentro de los candidatos para que entre al ensamble.

5.1.3. Asignación de Pesos

Ya que se eligió el m-ésimo miembro del ensamble, se debe determinar su peso α_m. De la Ecuación 11 se sigue que

$$\frac{\mathrm{d}E}{\mathrm{d}\alpha_m} = -W_c e^{-\alpha_m} + W_e e^{\alpha_m} \tag{14}$$

Igualando a cero y multiplicando por e^{α_m} se obtiene que

$$-W_c + W_e e^{2\alpha_m} = 0 \tag{15}$$

por lo que el óptimo se encuentra dado por

$$\alpha_m = \frac{1}{2} \ln\left(\frac{W_c}{W_e}\right) \tag{16}$$

Recordando que W es la suma total de los pesos, la Ecuación 16 se puede escribir como

$$\alpha_m = \frac{1}{2}\ln\left(\frac{W - W_e}{W_e}\right) = \frac{1}{2}\ln\left(\frac{1 - e_m}{e_m}\right) \tag{17}$$

donde $e_m = W_e/W$ representa la razón del error dados los pesos de todos los puntos del conjunto de datos.

5.1.4. Algoritmo

Para $m = 1$ hasta M:

1. De la lista de posibles clasificadores elegir al ϕ_m que minimice

$$W_e = \sum_{y \neq i\phi_m(\boldsymbol{x}_i)} w_i^{(m)}$$

2. Establecer el peso α_m del clasificador a

$$\alpha_m = \frac{1}{2}\ln\left(\frac{1 - e_m}{e_m}\right)$$

3. Actualizar los pesos de los datos para la siguiente iteración. Si $\phi(\boldsymbol{x}_i)$ es un fallo

$$w_i^{m+1} = w_i^m e^{\alpha_m} = w_i^m \sqrt{\frac{1 - e_m}{e_m}}$$

   Si es un acierto

$$w_i^{m+1} = w_i^m e^{-\alpha_m} = w_i^m \sqrt{\frac{e_m}{1 - e_m}}$$

4. Para la selección del conjunto de datos, AdaBoost tiene dos opciones:

   - Realizar un muestreo aleatorio considerando los pesos de cada dato, i.e., un muestreo con importancia.
   - Utilizar todo el conjunto de datos original, considerando los pesos como una manera de aumentar o disminuir el error del ensamble.

5. El algoritmo se detiene cuando el número de modelos base es alcanzado, o bien, cuando encuentra un modelo perfecto.

6. Para realizar las predicciones, AdaBoost simplemente computa las predicciones de todos los predictores y las pondera por medios de los pesos α_i. La clase predicha es aquella que recibe la mayoría de votos ponderados:

$$\hat{y}_i = \psi(\boldsymbol{x}_i) = \text{sign}\left(\sum_{j=1}^{M} \alpha_j \phi_j(\boldsymbol{x}_i)\right)$$

5.2. Gradient Boosting

Breiman menciona que en el caso de Boosting, lo que se hace en realidad es un proceso de optimización en una función de error, es decir, se busca minimizar el error generado por un clasificador de forma sucesiva. La idea anterior ya se exploró en AdaBoost considerando el error exponencial, el cual permita asignar pesos cada vez mayores tanto a los modelos de clasificación más relevantes como a los puntos del conjunto de datos más problemáticos para el ensamble. Además, debido a la naturaleza algorítmica inicial del modelo de Adaboost, los detalles de su rendimiento y propiedades fueron difíciles de analizar. Esto llevó a especular sobre las razones que originaban un bajo o alto rendimiento en diferentes situaciones. Para el desarrollo de esta sección, seguiremos la idea de Natekin y Knoll [11].

La formulación basada en el Gradiente Descendiente permitió establecer una conexión con el marco estadístico en el modelo, además de especificar las bases para desarrollar diferentes versiones de las *Gradient Boosting Machines*.

5.2.1. Estimación de una Función

Consideremos el problema clásico para estimar una función desde el punto de vista del aprendizaje supervisado. Se tiene un conjunto de datos formado por N puntos (x_i, y_i) y la meta es encontrar $\hat{f}$ que se acerque lo más posible a la función ideal f tal que $f : x \to y$. Para saber qué tan lejos la función estimada $\hat{f}$ se encuentra de f, se puede utilizar un función de perdida (o error) $\Psi(y, f)$ que debe ser minimizada:

$$\begin{aligned} \hat{f}(x) &= y \\ \hat{f}(x) &= \underset{f(x)}{\operatorname{argmin}} \quad \Psi(y, f(x)) \end{aligned} \tag{18}$$

Si se reescribe la estimación del problema en términos de valores esperados, la expresión resultante tendría que minimizar la función de pérdida esperada sobre la variable de salida $E_y\Psi[y, f(x)]$ condicionada a la variable de entrada respectiva x:

$$\hat{f}(x) = \underset{f(x)}{\operatorname{argmin}} \quad \underbrace{E_x[\overbrace{E_y\Psi[y, f(x)]}^{\text{Valor esperado de la pérdida de } y}|x]}_{\text{valor esperado sobre todo el conjunto de datos}} \tag{19}$$

Dado que hablamos de esperanza matemática, la variable y puede seguir diferentes distribuciones las cuales dependen de la especificación de la función del cálculo del error Ψ. Por ejemplo, si la salida del modelo es binaria $y \in \{0, 1\}$, es posible considerar como función de pérdida una distribución binomial. Si la salida del modelo es continua, se puede utilizar la medida L_2, el error absoluto o la pérdida de Huber, entre otras.

Una forma sencillas de abordar el problema de estimar funciones es limitar la búsqueda en un espacio de funciones paramétricas $f(x, \theta)$. Lo anterior nos obligaría a cambiar el problema de estimación de una función a uno de estimar parámetros:

$$\hat{f}(\boldsymbol{x}) = f(x, \hat{\theta}) \tag{20}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \quad E_x[E_y\Psi[y, f(x, \theta)]|x] \tag{21}$$

5.2.2. Optimización Numérica

En práctica, no es sencillo obtener soluciones analíticas para las estimaciones de los parámetros, por lo que se recurre a métodos iterativos. Dados M pasos iterativos, la estimación de los parámetros se puede escribir como una estimación por pasos:

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i \tag{22}$$

La forma más sencilla para realizar esta estimación es por medio de Gradiente Descendiente. Dados N puntos $(\boldsymbol{x}_i, y_i)$, se desea minimizar el error $J(\theta)$ considerando los datos observados:

$$J(\theta) = \sum_{i=1}^{N} \Psi(y_i, f(\boldsymbol{x}_i, \hat{\theta})) \tag{23}$$

Noten que no se establece una función de error o pérdida en específico. Se puede considerar cualquiera que se proponga.

La idea del gradiente descendiente es minimizar una función $f(\boldsymbol{x})$ considerando el gradiente. Para $f\colon \mathbb{R}^n \to \mathbb{R}$ su gradiente $\nabla f\colon \mathbb{R}^n \to \mathbb{R}^n$ en un punto $\boldsymbol{p} = (x_1, \ldots, x_n)$ se define como:

$$\nabla f(\boldsymbol{p}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{p}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\boldsymbol{p}) \end{bmatrix}.$$

La idea principal del gradiente descendiente es llegar al mínimo local de una función yendo en la dirección contraria del gradiente, que indica la dirección con mayor pendiente en la función. El algoritmo iterativa plantea dar pequeños pasos en dicha dirección de la siguiente manera:

$$\boldsymbol{p}_{m+1} = \boldsymbol{p}_m - \eta \nabla f(\boldsymbol{p}_m) \tag{24}$$

- El parámetro η permite escalar el valor del gradiente, lo que hace cada paso más grande o más pequeño.
- En Machine Learning, η es la razón de aprendizaje (*learning rate*).
  - Si es muy pequeño, tarda más en converger.
  - Si es muy grande, da saltos grandes, inclusive no llegando a converger.

Con esta idea, el Gradiente Descendiente empleado en Gradient Boosting se ve de la siguiente manera:

1. Inicializar los parámetros $\hat{\theta}_0$.
2. Obtener el parámetro $\hat{\theta}^t$ que recopila los valores anteriores:

$$\hat{\theta}^t = \sum_{i=0}^{t-1} \hat{\theta}_i$$

3. Evaluar el gradiente de la función de pérdida $\nabla J(\theta)$ dados los parámetros estimados del ensamble:

$$\nabla J(\theta) = \{\nabla J(\theta_i)\} = \left[\frac{\partial J(\theta)}{\partial J(\theta_i)}\right]_{\theta=\hat{\theta}^t}$$

4. Calcular el nuevo incremento de $\hat{\theta}^t$:

$$\hat{\theta}_t \leftarrow -\nabla J(\theta)$$

5. Añadir el nuevo estimado de $\hat{\theta}_t$ al ensamble.

5.2.3. Optimización en el Espacio de Funciones

La principal diferencia entre los métodos de boosting y cualquier otro modelo de aprendizaje automático es que la optimización se realiza en un espacio de funciones. Se parametriza el estimado de la función $\hat{f}$ en su forma de función aditiva:

$$\hat{f}(\boldsymbol{x}) = \hat{f}^M(\boldsymbol{x}) = \sum_{i=0}^{M} \hat{f}_i(\boldsymbol{x}) \tag{25}$$

De esta manera, M representa el número de iteraciones donde $\hat{f}_0(\boldsymbol{x})$ es el valor inicial y $\{\hat{f}_i\}$ son los incrementos, también llamados *boosts*.

Ahora, ¿cuáles son los parámetros θ En este caso, son los modelos base de aprendizaje $h(\boldsymbol{x}, \theta)$, los cuales suelen ser Árboles de Decisión. Con esto, es posible formular la idea para incrementar funciones por medio de los modelos base. Considerando el algoritmo de gradiente descendiente explicado anteriormente, es necesario especificar el parámetro η. Para estimar la función, la regla de optimización se define de la siguiente forma:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \eta_t h(\boldsymbol{x}, \theta_t) \tag{26}$$

$$(\eta_t, \theta_t) = \underset{\eta,\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \psi(y_i, \hat{f}_{t-1} + \eta h(\boldsymbol{x}_i, \theta) \tag{27}$$

El algoritmo se puede apreciar en la Fig. (4).

6. Bosques Aleatorios

En la sección 3 es abordó el tema de Bagging como un modelo de ensamble, el cual permite reducir la varianza estimada de una función predictora. Esta condición se ajusta a los árboles de decisión, los cuales presentan alta varianza y bajo sesgo. Los árboles aleatorios propuestos por Breiman [3] son una modificación al modelo de bagging ya que construye una colección de árboles sin correlación para después promediarlos. En práctica, su desempeño es similar al bagging, pero son más sencillos de entrenar y ajustar. La idea de esta sección sigue la de Hastie et al. [8].

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

1: initialize $\widehat{f}_0$ with a constant
2: **for** $t = 1$ to M **do**
3: compute the negative gradient $g_t(x)$
4: fit a new base-learner function $h(x, \theta_t)$
5: find the best gradient descent step-size ρ_t:
$\rho_t = \arg\min_\rho \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right]$
6: update the function estimate:
$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$
7: **end for**

Figura 4: Algoritmo de Gradiente Boosting propuesto por Friedman [7].

6.1. Definición

La idea del Bagging es promediar modelos sin (o bajo) sesgo pero ruidosos para reducir su varianza. Los Árboles de Decisión son ideales para esta tarea ya que son capaces de capturar información compleja en la estructura de la información. Además, conforme crecen, el sesgo se reduce.

Al momento de generar los árboles en el ensamble (Bagging, en este caso) estos son idénticamente distribuidos, lo que ocasiona que valor esperado del promedio de B árboles es el misma que el valor esperado de cada uno de ellos. Esto implica que el sesgo presente en el Bagging es el mismo que los árboles individualmente y lo único que se se puede reducir es la varianza. ¿Qué diferencia existe con el Boosting? Al generarse de forma iterativa, no siguen la misma distribución.

Un promedio de B v.a.i.i.d, cada una con varianza σ^2 , tiene varianza $\frac{1}{B}\sigma^2$. Si las variables son i.d. con correlación positiva ρ, la varianza del promedio esta dada por

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{28}$$

Conforme B aumenta, el segundo término de la expresión va a cero, por lo que la correlación entre pares de árboles en el Bagging afecta el beneficio del promedio, i.e., no reduce la varianza.

La idea de los Bosques Aleatorios yace en mejorar la reducción de la varianza propuesta por Bagging al reducir la correlación entre los árboles. Lo anterior se logra al momento de crecer los árboles eligiendo de forma aleatoria los valores de entrada. Después de realizar el Bootstrap, se seleccionan $m \leq p$ del total de las variables de entrada al azar como candidatos para el corte de los árboles. Algunos valores populares para m son $\sqrt{p}$ o 1.

Después de que B árboles $\{T(\boldsymbol{x}, \Theta_b)\}_{b=1}^{B}$ son creados, el Bosque Aleatorio para el caso de regresión se encuentra dado por

$$\hat{f}^B(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} T(\boldsymbol{x}, \Theta_b) \tag{29}$$

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Figura 5: Algoritmo para bosques aleatorios.

donde Θ_b caracteriza el b-ésimo árbol del Bosque Aleatorio: valores de corte, cortes en cada nodo y los valores de los nodos terminales. Reducir el valor de m reducirá el valor de la correlación entre cualquier pareja de árboles del ensamble y, considerando la Ecuación 28, el valor de la varianza disminuirá. El algoritmo de bosques aleatorios se encuentra en la Fig. (5).

6.2. Detalles de Implementación

Cuando se usa para clasificación, el Bosque Aleatorio obtiene un voto de cada árbol, los cuales se someten a un proceso de votación. Cuando se utiliza para regresión, las predicciones de cada árbol se suman y se promedian como en la Ecuación 29. Breiman añade las siguientes recomendaciones:

- Para clasificación, el valor de m es $\lfloor\sqrt{p}\rfloor$ y el tamaño mínimo de los nodos es uno.
- Para regresión, el valor de m es $\lfloor p/3\rfloor$.

Otro elemento importante en los bosques aleatorios son las muestras fuera de la bolsa. Para cada observación o dato $z_i(\boldsymbol{x}_i, y_i)$, se construyen un bosque aleatorio promediando únicamente los árboles correspondientes a las muestras de bootstrap en donde z_i no aparece. En particular, las muestras fuera de la bolsa se utilizan para calcular el error del bosque aleatorio, simular a la validación cruzada con N pliegues.

7. Ejercicios

1. En la sección de clasificadores votantes, se menciono que después de 1000 lanzamientos la probabilidad de que la mayoría de ellos sean cara es cercana al 75 %. Vamos a realizar un pequeño experimento para explorar este resultado en la computadora:

a) Simular 1000 lanzamientos de una moneda con dichos pesos (51/49) para cara y cruz, respectivamente. Determinar la mayoría resultante en el experimento.

b) Repetir el experimento anterior 10,000 veces para determinar la *probabilidad* de obtener una mayoría de caras con 1000 lanzamientos.

c) Repetir el experimento aumentando el número de lanzamientos desde 500 hasta 10,000 en incrementos de 10. Graficar la *probabilidad* de obtener una mayoría de caras contra el número de lanzamientos.

2. Un método de ensambles que no cubrimos en esta parte es el de Stacking [15]. Investiguen, expliquen y resuman la idea general de este método de ensambles. Agregar un diagrama que explique su funcionamiento.

3. Crear un modelo de ensamble basado en Stacking usando como conjunto de datos el dataset MNIST en Scikit-Learn. Como modelos base usen un Bosque Aleatorio, una MVS con kernel RBF y una modelo de regresión logística. Determinar los rendimientos (usar como medida la weighted F1 score) de cada modelo individualmente, y compararlos con el rendimiento del ensamble.

8. Actualizaciones

12/2023 Actualización de las Figuras 1 y 2. Corrección de errores ortográficos y modificación de estilo. Se añadieron nuevos ejercicios.

01/2023 Se añadió la sección de GradientBoosting y Bosques Aleatorios.

01/2022 Primera versión del documento.

Referencias

[1] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[2] Leo Breiman. Pasting small votes for classificationin large databases and on-line. *Machine Learning*, 36:85–103, 1999.

[3] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[4] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, page 1–15, Berlin, Heidelberg, 2000. Springer-Verlag.

[5] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

[6] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, aug 1997.

[7] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, New York, NY, 2 edition, 2012.

[9] Cheng Li. A gentle introduction to gradient boosting. Technical report, Northeastern University.

[10] Gilles Louppe. Understanding random forests: From theory to practice. July 2014.

[11] Alexey Natekin and Alois Knoll. Gradient boosting machines,a tutorial. *Frontiers in Neurorobotics*, 7, 2013.

[12] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[13] Matteo Re and Giorgio Valentini. *Ensemble methods: A review*, pages 563–594. 01 2012.

[14] Raul Rojas. Adaboost and the super bowl of classiers a tutorial introduction to adaptive boosting. Technical report, Freie Universitat Berlin, 2009.

[15] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[16] Zhi-Hua Zhou. *Ensemble Learning*, pages 270–273. Springer US, Boston, MA, 2009.